Breviar

Laborator 2 - Operații cu șiruri de caractere

Breviar 2 - Utilizare clase Java

Operații cu șiruri de caractere - String, StringBuffer

Introducere

În limbajele de programare, majoritatea tipurilor de date utilizate pentru variabile sunt **valorile booleene**, **șirurile de caractere** (sau vectori de caractere terminati cu `\0`) și **valorile numerice**. În contrast cu limbajele **C** sau **C++**, în Java modul de gestiune a șirurilor de caractere este unul diferit, deoarece:

- fiecare `char` reprezintă o valoare **Unicode** pe `16 biți` și nu `1 octet` (ca în **C** / **C++**);
- valorile de tip șiruri de caractere sunt gestionate de obiecte **String**;
- sintaxa vă permite să utilizați **String** asemenea unui tip primitiv de date (puteți folosi operatorul = pentru a le inițializa);
- **String** sunt obiecte **imutabile** (**immutable**), în sensul că odată ce sunt create, ele nu își pot schimba valoarea.

Clasa String

În Java, **String** este o clasă elementară care oferă funcțiile de bază pentru manipularea șirurilor de caractere. Ca utilizare, această clasă se folosește pentru compararea a două șiruri de caractere, extragerea de subșiruri, determinarea lungimii unui șir sau alte operații care vor fi descrise mai jos.

După cum vom vedea, orice șir este, de fapt, o instanță a clasei **String** definită în pachetul **java.lang**.

Instanțierea unui obiect String

```
class Test {
    String str1 = "text";
    String str2 = new String("text");
    char data[] = {'t', 'e', 'x', 't'};
    String str3 = new String(data);
}
```

În Java, **literalele** sunt numerele, textul și alte informații care reprezintă direct o valoare.

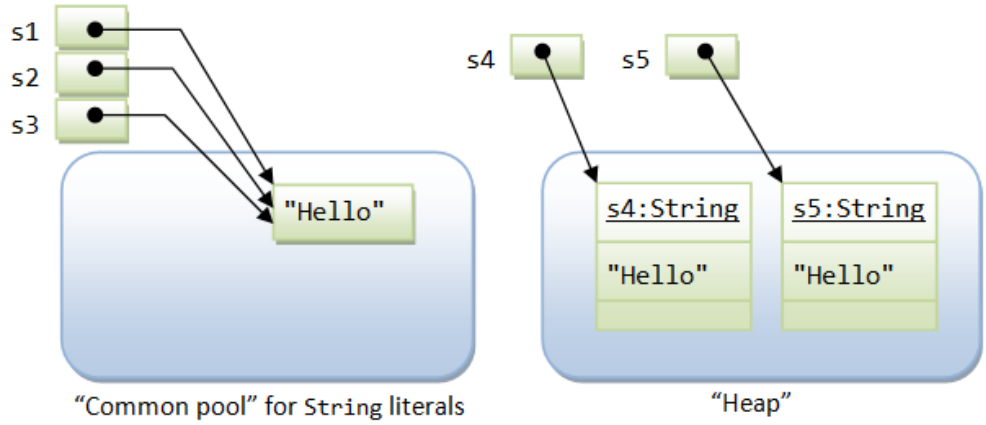
Literalele șir constau dintr-o serie de caractere încadrate între ghilimele duble:

```
String autor="Liviu Rebreanu";
```

Atunci când se folosește un **literal șir** în Java, se stocheaza valoarea sa ca obiect `String`. Nu trebuie să instanțiem explicit un nou obiect - ca în cazul celorlalte obiecte cu care vom lucra - deci este la fel de ușor de lucrat cu acest tip ca în cazul tipurilor primitive.

Șirurile sunt diferite în ceea ce privește acest aspect - nici unul dintre tipurile primitive nu sunt pastrate ca obiecte atunci când sunt folosite.

```
String s1 = "Hello";              // String literal
String s2 = "Hello";              // String literal
String s3 = s1;                   // same reference
String s4 = new String("Hello");  // String object
String s5 = new String("Hello");  // String object
```



Operații uzuale cu obiecte de tip String

```
class Test {
    public static void main(String args[]) {
        String str = "Laborator POO";
        String str1 = " 2018-2019";
        String str2 = "POO";
        String str3;
        char c = str.charAt(1); //caracterul de pe pozitia 1
        str3 = str.concat(str1); //concatenarea de siruri
        System.out.println(str3 + " != " + str);
        if(str.equals(str1)) {
            System.out.println("Sirurile sunt egale");
        }
        //pozitia de inceput a subsirului str2 in sirul str1
        int poz = str.indexOf(str2);
        int lungime = str.length();
        //operatorul + introduce posibilitatea concatenarii
        str3 = str + str1;
    }
}
```

Compararea a două șiruri de caractere

Pentru a compara două șiruri de caractere, reprezentate prin obiecte de tip **String**, putem folosi metoda `equals`.

Dacă vom folosi operatorul ==, atunci vom compara referințele.

Pentru a înțelege mai bine cum stau lucrurile, puteți analiza exemplul următor.

```
// Declarațiile
String s1 = "Hello";              // String literal
String s2 = "Hello";              // String literal
String s3 = s1;                   // same reference
String s4 = new String("Hello");  // String object
String s5 = new String("Hello");  // String object

// Compararea string-urilor
s1 == s1;         // true, same pointer
s1 == s2;         // true, s1 and s1 share storage in common pool
s1 == s3;         // true, s3 is assigned same pointer as s1
s1.equals(s3);    // true, same contents
```

```
s1 == s4;         // false, different pointers
s1.equals(s4);    // true, same contents
s4 == s5;         // false, different pointers in heap
s4.equals(s5);    // true, same contents
```

În Java, operatorul de concatenare + este extrem de flexibil, în sensul că permite concatenarea sirurilor cu obiecte de orice tip care au o reprezentare de tip sir de caractere. Mai jos, sunt prezentate câteva exemple pentru a lămuri ceea ce execută compilatorul atunci când întâlneste o secventă care contine operatorul de concatenare.

```
class Test {
    public static void main(String args[]) {
        String x = "a" + 1 + "b";
        x = 1 + 2 + "c" + 1 + 2;
        x = "" + 1 + 2 + "c";
        x = 1.5 + " " + 2.3;
    }
}
```

Atenție însă la ordinea de efectuare a operațiilor. Șirul **s** va avea valoarea **"3a12"**, primul + fiind operatorul matematic de adunare, iar al doilea + este cel de concatenare a șirurilor.

```
s = 1 + 2 + "a" + 1 + 2;
```

Clasa StringBuffer

În Java, un șir de caractere poate fi reprezentat printr-un vector format din elemente de tip (`char`), un obiect de tip **String** sau un obiect de tip **StringBuffer**.

Dacă un șir de caractere este constant (nu se dorește schimbarea conținutului său pe parcursul execuției programului), atunci el va fi declarat de tipul **String**, această clasă fiind nemutabilă, altfel, va fi declarat de tip **StringBuffer**, deoarece această clasă este mutabilă.

Diferența principală între aceste clase este că **StringBuffer** pune la dispoziție metode pentru modificarea conținutului șirului, cum ar fi: `append`, `insert`, `delete`, `reverse`.

Uzual, cea mai folosită modalitate de a lucra cu șiruri este prin intermediul clasei **String**, care are și unele particularități față de restul claselor menite să simplifice cât mai mult folosirea șirurilor de caractere. Clasa **StringBuffer** va fi utilizată predominant în aplicații dedicate procesării textelor cum ar fi editoarele de texte.

Instanțierea unui obiect StringBuffer

```
StringBuffer str = new StringBuffer("abc");
StringBuffer str1 = new StringBuffer();
```

Operații uzuale cu obiecte de tip StringBuffer

```
class Test {
    public static void main(String args[]) {
        StringBuffer str = new StringBuffer("abc");
        StringBuffer str1 = new StringBuffer();
        //se adauga str la sbuf
        str1.append(str);
        System.out.println("str1 = " + str1);
        //se adauga reprezentarea ca sir de caractere a nr 2
        str1.append(1);
        str1.append("xyz");
        //se insereaza sirul "bc" in str1
        str1.append("abc", 1, 3);
        System.out.println(str1);
        //se inlocuiesc caracterele de la pozitiile 1-3
        //cu sirul s => aPOO1xyzbc
```

```
        String s = "POO";
        str1.replace(1, 3, s);
        //se sterg caracterele de la pozitiile 1-3
        str1.delete(1, 3);
    }
}
```

Clasa Vector

Un vector este o structura de date dinamica in Java. Se pot adauga elemente in orice pozitie, se pot sterge elemente, se pot inlocui elemente, se poate determina pozitia pe care se afla un element, se poate determina numarul de elemente dintr-un vector. In Java, un obiect de tip **Vector** lucreaza cu elemente de tip **Object**, aceasta insemnand ca permite stocarea oricaror tipuri de elemente. Pentru o intelegere mai buna a importantei utilizarii acestei clase, este recomandata parcurgerea exemplului de mai jos.

```
public class Exemplu {
    public static void main(String args[]) {

        //Instantierea obiectului
        Vector vect = new Vector();

        //Adaugarea in vector a unor numere intregi
        vect.add(20);
        vect.add(5);
        vect.add(7);

        //Determinarea pozitie lui 5
        System.out.println(vect.indexOf(5));

        //Determinarea primului element
        System.out.println(vect.firstElement());

        //Modificarea elementului de pe pozitia 2
        vect.set(2, 10);

        //Determinarea primului element
        int first = (int) vect.get(0);

        //Afisarea vectorului - recomandat
        System.out.println(vect);
        //Afisarea vectorului - **NERECOMANDAT** !!
        for(int i = 0 ; i < vect.size(); i++) {
            System.out.println(vect.get(i));
        }
    }
}
```

Interviu

Această secțiune este una opțională și încearcă să vă familiarizeze cu o serie de întrebări ce pot fi adresate în cadrul unui interviu tehnic. De asemenea, această secțiune poate fi utilă și în pregătirea pentru examenul final de la această disciplină.

Întrebări interviu

1. De ce este considerată clasa **String** imutabilă?
2. De ce este considerată clasa **StringBuffer** mutabilă?
3. Care este diferenta esentială între **StringBuffer** si **StringBuilder**?
4. Se pot compara două obiecte de tip **String**? Argumentați!
5. De ce un vector de caractere este mai potrivit decât un obiect de tip **String** pentru retinerea unei parole?
6. Ce se întâmplă atunci când folosim operatorul == pentru a compara obiecte de tip **String**?

7. Există vreo modalitate prin care un obiect se poate converti în obiect de tip **String**?

Feedback

Pentru îmbunatătirea constantă a acestui laborator, vă rog să completati formularul de feedback disponibil aici [https://docs.google.com/forms/d/1Pg4lBVNOyx1MZp8o45wFHYUd089vL8YfExuHHsuZO9Q/viewform?c=0&w=1].

De asemenea, vă rog să semnalati orice greseală / neclaritate depistată în laborator pentru a o corecta. Vă multumesc anticipat!

poo/breviare/breviar-02.txt · Last modified: 2018/10/01 13:21 by carmen.odubasteanu